计算机科学 1999Vol. 26№. 4

数据仓库上的 Gcube 操作*)

The Gcube Operator on Data Warehouses

高 宏 李建中
(黑龙江大学 哈尔滨 150080)

Abstract The Cube operator in the on-line analytical processing (OLAP) is an operator in common use, but resently, the definition of the Cube operator and its semantic possessed very large limitations. This paper analyzes the disadvantages of the Cube operator and proposes a new operator, Gcube. Gcube is an extension of the Cube operator which overcomes the disadvantages of the Cube operator. The algorithm and optimization techniques for implementing the Gcube operator are presented in the paper also.

Keywords Data warehouse, OLAP, Cube, Gcube

1 问题的提出

商业和企业界认为,数据仓库上的联机分析处理(OLAP)应用是决策分析的关键。支持 OLAP 应用的多维数据模型[1]正在得到人们的普遍接受。在多维数据模型中,数据记录中的某些域或属性被选作度量属性,其它域或属性被称为维属性或函数属性。在多维数据库中,具有相同函数属性值的数据记录的度量属性值集合组合为一个聚集值(通过聚集函数)。多维数据库可以视为由维属性值索引的多维数组,每个数组单元存储相应的维属性值组合所对应的度量属性值集合的聚集值。以后,在不引起混淆的情况下,我们把多维数据库简单地称为关系。多维数据库通常也称为 Cube。人们把产生多维数据库的过程称为 Cube 操作。文献[2]和[3]给出了 Cube 操作的严格定义。Cube 操作是 SQL 语言中 Group By 操作的多维扩充。设 R 是一个维属性集合为 $\{a_1,\cdots,a_n\}$、度量属性集合为 $\{b_1,\cdots,b_t\}$ 的关系,对于 $1\leqslant i\leqslant k$,$agg_i(m)$ 是一个定义在度量属性集 m 上的聚集函数(如 *Sum*、*Max* 等)。R 的 Cube 操作的定义为

SELECT $a_{i_1},\cdots,a_{i_k},agg_{j_1}(m_1),agg_{j_2}(m_2),\cdots,agg_{j_s}(m_s)$
FROM R
CUBE BY $a_{i_1},\cdots,a_{i_k}$,

其中,$m_i\subseteq\{b_1,\cdots,b_t\}(1\leqslant i\leqslant j)$,$\{a_{i_1},\cdots,a_{i_k}\}\subseteq\{a_1,\cdots,a_n\}$ 称为 CUBE BY 属性。这个 Cube 操作将产生 2^k 个类似于 SQL 中的 GROUP BY 操作,每个操作的 GROUP BY 属性集合是 $\{a_{i_1},\cdots,a_{i_k}\}$ 的一个子集合,每个这样的 GROUP BY 操作结果称为 Cube 操作的一个 *cuboid*。Cube 操作建立一个表或关系 C。C 的属性集合为 $\{a_{i_1},\cdots,a_{i_k},AGG_1,\cdots,AGG_s\}$,$AGG_1,\cdots,AGG_s$ 对应于 $agg_{j_1}(m_1),\cdots,agg_{j_s}(m_s)$。如果 C 的元组 T 是 GROUP BY 属性集合为 S 的 *cuboid* 之元组,则 T 在 $\{a_{i_1},\cdots,a_{i_k}\}-S$ 属性上的值为一个表示"被聚集"的特殊值"*all*"。T 在 $AGG_1,\cdots,AGG_s$ 上的值分别是 $agg_{j_1}(m_1),\cdots,agg_{j_s}(m_s)$。例如,我们有一个关系 *trans*(*StoreID*, *ItemID*, *Date*, *Quantity*, *Price*),其实例如图 1 所示。*trans* 上的 Cube 操作:

```
SELECT StoreID, ItemID, Date, Sum(Quantity)
FROM trans
CUBE BY StoreID, ItemID, Date
```

StoreID	ItemID	Date	Quantity	Buyer	Price
S1	I1	1997.08.01	15	B1	10
S1	I2	1997.08.01	20	B2	11
S1	I1	1997.08.01	12	B3	10
S2	I1	1997.08.01	13	B4	12
S2	I2	1997.08.01	11	B5	14
S2	I2	1997.08.01	5	B6	14

图 1

*)本项研究工作得到国家八六三计划基金资助。

的结果如图 2 所示。上述 Cube 操作等价于如下的扩充 SQL 语句：

```
SELECT StoreID, ItemID, Date, Sum(Quantity)
FROM trans
GROUP BY StoreID, ItemID, Date
UNION
SELECT StoreID, ItemID, all, Sum(Quantity)
FROM trans
GROUP BY StoreID, ItemID
UNION
SELECT StoreID, all, Date, Sum(Quantity)
FROM trans
GROUP BY StoreID, Date
UNION
SELECT all, ItemID, Date, Sum(Quantity)
FROM trans
GROUP BY ItemID, Date
UNION
SELECT StoreID, all, all, Sum(Quantity)
FROM trans
GROUP BY StoreID
UNION
SELECT all, ItemID, all, Sum(Quantity)
FROM trans
GROUP BY ItemID
UNION
SELECT all, all, Date, Sum(Quantity)
FROM trans
GROUP BY Date
UNION
SELECT all, all, all, Sum(Quantity)
FROM trans.
```

StoreID	ItemID	Date	Sum(Quantity)
S1	I1	1997.08.01	27
S1	I2	1997.08.01	20
S2	I1	1997.08.01	13
S2	I2	1997.08.01	16
S1	I1	all	27
S1	I2	all	20
S2	I1	all	13
S2	I2	all	16
S1	all	1997.08.01	47
S2	all	1997.08.01	29
all	I1	1997.08.01	40
all	I2	1997.08.01	36
S1	all	all	47
S2	all	all	29
all	I1	all	40
all	I2	all	36
all	all	1997.08.01	76
all	all	all	76

图 2

以下，我们用 *ALL* 表示 GROUP BY 属性集合为空集合的 *cuboid*。显然，对于任意维属性子集合 $\{B_1,\cdots,B_l\}$，基于 $\{B_1,\cdots,B_l\}$ 的 Cube 是 $\{B_1,\cdots,B_l\}$ 的所有子集上的 *cuboid* 的并集。因此，为了计算基于 $\{B_1,\cdots,B_l\}$ 的 Cube，我们需要计算组成这个 Cube 的所有 *cuboid*。

到目前为止，有关 Cube 的研究工作都基于上述定义。上述 Cube 操作的定义具有很大的局限性而且不完备，不能适应很多 OLAP 应用的要求。为了说明这些问题，我们来考察下面两个例子。在下面的例子中，我们使用一个有关零售商店的数据仓库，其中包括如下关系：

trans(*StoreID*, *ItemID*, *Date*, *Quantity*, *Price*),
stores(*StoreID*, *City*, *Region*),
items(*ItemID*, *Name*, *Category*, *Cost*)

例 1 查询各个零售商店每种具体商品的销售量、各类商品的销售量、每类商品在所有零售商店的总销售量。使用 Cube 操作，我们需要分两步实现该查询：

步骤 1：

```
SELECT StoreID, ItemID, Category, Quantity INTO temp
FROM trans, items
WHERE trans. ItemID=items. ItemID;
```

步骤 2：

```
SELECT StoreID, ItemID, Category, SUM(Quantity)
FROM temp
CUBE BY StoreID, ItemID, Category.
```

由于 Cube 只对单一关系执行聚集，所以在执行 Cube 操作前，步骤 1 首先对关系 *trans* 和 *items* 执行连接操作。步骤 2 的 Cube 操作计算了基于 $\{StoreID, ItemID, Category\}$ 的所有八个 *cuboid*：*StoreIDItemIDCategory*，*StoreIDItemID*，*StoreIDCategory*，*ItemIDCategory*，*StoreID*，*ItemID*，*Category* 和 *ALL*，而用户只需要计算 *StoreIDItemIDCategory*，*Category* 和 *StoreIDCategory* 三个 *cuboid*。显然，上述 Cube 操作降低了查询效率。

例 2 计算 96 年以来东北地区各城市各种电器产品的销售量、最低售价以及每一城市的总销售量。使用 Cube 操作，我们需要分两步实现该查询：

步骤 1：

```
SELECT ItemID, City, Date, Quantity, Price INTO temp
FROM trans, stores, items
WHERE stores. Region="东北"AND
      items. Category="电器"AND
      trans. Date>=1996.01.01 AND
      trans. StoreID=stores. StoreID AND
      trans. ItemID=items. ItemID;
```

步骤 2：

```
SELECT ItemID, City, Date, SUM(Quantity), MIN
(Price)
FROM temp
CUBE BY ItemID, City, Date.
```

由于 Cube 只对单一关系执行聚集，所以在执行 Cube 操作前，步骤 1 首先对相关的三个关系执行连接，然后步骤 2 执行 Cube 操作。Cube 操作的结果是对 $\{ItemID, City, Date\}$ 的所有 *cuboid* 执行了相同的聚集操作 *SUM*() 和 *MIN*()。而实际上，用户

并没有要求对 *cuboid City* 执行 *MIN*()运算。从上面的例子我们可以看到 Cube 操作存在以下问题：

(1)操作关系只能为一个关系，而在实际应用中，可能要对多个关系的连接执行 Cube 操作，亦即 Cube 中的属性来自不同的关系。

(2)对组成 Cube 的每个 *cuboid* 都使用相同的聚集函数，而实际应用可能要求不同 *cuboid* 使用不同的聚集函数。

(3)在许多数据分析应用中，对多维数据进行聚集计算时，用户往往只关注于 Cube 中某些 *cuboid* 的结果，忽略其他的 *cuboid*。例如，决策者只想了解各类商品在各个城市的全年销售情况，此时，系统无需计算所有整个 Cube，只计算用户感兴趣的那些 *cuboid*。Cube 操作不能满足这种要求，它只能计算所有的 *cuboid*。

为了解决上述问题，本文对 Cube 进行了扩充，提出了一般化的 Cube 操作，称为 Gcube。Gcube 操作克服了 Cube 操作的局限性，避免了上述三个问题，具有更高的适应性。本文还研究了 Gcube 操作的实现问题，提出了实现 Gcube 操作的有效方法和优化处理技术。

2 Gcube 操作

在定义 Gcube 操作之前，我们需要定义子 Cube(简记作 SubCube)的概念。

定义 1 设有一个由 n 个维属性组成的 Cube，包含有 2^n 个 *cuboid*。这一组 *cuboid* 的任意一个子集称为一个子 Cube，简记作 SubCube。

现在我们来定义 Gcube 操作。Gcube 是 Cube 操作的扩充。我们首先给出 Gcube 操作的语法定义，然后对其语义加以说明。在语法定义中，〈X〉表示 X 是一个语法变量。[X]表示 X 是可选项，即 X 可有可无。{X}$_0^n$ 表示 X 可以出现任意次，包括 0 次。Gcube 操作的语法定义如下：

SELECT 〈*dimension_attribute_list*〉,〈*Agg_function*〉(〈*measures_attribute_list*〉){,〈*Agg_function*〉(〈*measures_attribute_list*〉)}$_0^n$[INTO 〈*relation_name*〉]
FROM 〈*relation_name*〉{,〈*relation_name*〉}$_0^n$
[WHERE 〈*condition*〉]
[Gcube BY [〈*Agg_function*〉{,〈*Agg_function*〉}$_0^n$]]
[SubCube BY
〈*dimension_attribute*〉{〈*dimension_attribute*〉}$_0^n$ {,〈*dimension_attribute*〉{〈*dimension_attribute*〉}$_0^n$}$_0^n$
:〈*Agg_function*〉{,〈*Agg_function*〉}$_0^n$
{;〈*dimension_attribute*〉{〈*dimension_attribute*〉}$_0^n$ {,〈*dimension_attribute*〉{〈*dimension_attribute*〉}$_0^n$}$_0^n$
:〈*Agg_function*〉{,〈*Agg_function*〉}$_0^n$}$_0^n$].

其中：〈*dimension_attribute_list*〉表示维属性表，称为 GCUBE BY 属性集合；{*Agg_function*)表示聚集函数名；〈*measures_attribute_list*〉表示度量属性表；〈*relation_name*〉表示关系名；〈*condition*〉表示 SQL 语言的条件表达式；〈*dimension_attribute*〉表示维属性。

Gcube 操作的语义如下：

(1)Gcube 操作的 SELECT 子句中的〈*dimension_attribute_list*〉指定了一组属性，称为 GCUBE BY 属性集合。该 Gcube 操作就是要计算〈*dimension_attribute_list*〉指定的 GCUBE BY 属性集合上的全部或部分 *cuboid*。SELECT 子句中的〈*Agg_function*〉(〈*measures_attribute_list*〉){,〈*Agg_function*〉(〈*measures_attribute_list*〉)}$_0^n$ 说明了 Gcube 操作在计算 *cuboid* 时需要使用的一组定义在度量属性集合上的聚集函数。

(2)Gcube 操作的 INTO 〈*relation_name*〉子句表示 Gcube 操作的结果存储到关系〈*relation_name*〉。如果 Gcube 操作中不包括 INTO 子句，则在终端显示器上显示操作结果。

(3) FROM 〈*relation_name*〉[{,〈*relation_name*〉}$_0^n$]子句给定了 Gcube 操作的一组操作关系。

(4)Gcube BY 子句具有两种形式。第一种形式为 Gcube BY 〈*Agg_function*〉{,〈*Agg_function*〉}$_0^n$。表示使用指定的聚集函数集合(*Agg_function*){,(*Agg_function*)}$_0^n$ 计算所有的 *cuboid*。第二种形式为 Gcube BY，表示使用 SELECT 子句中的所有聚集函数计算所有的 *cuboid*。

(5)如果 Gcube 操作中无 Gcube BY 子句，则 SubCube BY 说明了需要而且仅需要计算的 *cuboid* 以及计算各 *cuboid* 所使用的聚集函数；否则 SubCube BY 补充说明某些 *cuboid* 还需要使用在 SELECT 子句中出现但是在 Gcube BY 子句中未出现的一些聚集函数。

显然，如果一个 Gcube 操作中既没有 Gcube BY 子句也没有 SubCube 子句，则该操作是一个典型的 SQL 语句。

下面我们以上节的数据仓库为例，说明 Gcube 操作的用法和优点。

例 3 查询各个零售商店每种具体商品的销售量、各类商品的销售量、每类商品在所有零售商店的总销售量。使用 Gcube 操作，我们可以很容易地实现这个查询：

SELECT *StoreID*,*ItemID*,*Category*,*SUM*(*Quantity*)
FROM *trans*,*items*
SubCube BY *StoreIDItemIDCategory*, *StoreIDCategory*, *Category*;*SUM*.

例 4 计算 96 年以来东北地区各城市的各种

电器产品销售量、最低售价以及每一城市的总销售量。使用 Gcube 操作，我们只需一个操作就可以实现这个查询：

```
SELECT ItemID, City, Date, SUM(Quantity), MIN(Price)
FROM trans, stores, items
WHERE stores.Region="东北"AND
      items.Category="电器"AND
      trans.Date>=1996.01.01
SubCube BY ItemIDCity:SUM,MIN;City:SUM
```

例 5 查询每种商品在各零售商店的销售量、最低售价、每种商品的总销售量和每一零售商店的总销售量。

```
SELECT StoreID,ItemID,SUM(Quantity),MIN(Price)
FROM trans
Gcube BY SUM
SubCube BY StoreIDItemID:MIN
```

从上面的例子和 Gcube 操作的定义可以看出，Gcube 操作解决了 Cube 操作存在的问题，具有如下优点：

(1)可以对多个关系的连接作 Cube，亦即 Cube 中的属性可以来自不同的关系。

(2)允许对操作关系进行选择操作，在满足条件的原始关系的子集合上执行 Gcube 操作。

(3)即可以计算对应于 GCUBE BY 属性集合的所有 *cuboid*，也可以选择计算特定的 *cuboid* 集合，而忽略其他的 *cuboid*。

(4)既可以在计算所有的 *cuboid* 时使用相同的聚集函数集合，也可以使不同的 *cuboid* 使用不同的聚集函数集合来计算。

3 Gcube 操作的实现

本节首先给出一个实现 Gcube 操作的简单算法，然后讨论实现 Gcube 操作的优化技术。给定一个 Gcube 操作 GCQ，我们可以使用如下的 Simple_Gcube 算法实现 GCQ：

(1)把 GCQ 分解为两段，第一段是 GCQ 中从 SELECT 子句到 WHERE 子句部分，称为 SQL 查询段。第二段由 GCQ 的 Gcube BY 和 SubCube BY 子句构成，称为 GCUBE 操作段。

(2)使用 SQL 查询优化处理方法优化处理 GCQ 的 SQL 查询段，结果存储到临时关系 TMP。

(3)在 TMP 上处理 GCQ 的 Gcube 操作段：A)若存在 Gcube BY 选项，则使用 SQL 的 GROUP BY 优化处理方法和相应的聚集函数计算 SELECT 子句中给出的 GCUBE BY 属性集合对应的每个 *cuboid*；B)若存在 SubCube BY 选项，则表示计算 SELECT 子句给定的 GCUBE BY 属性集合上的一组特定的 *cuboid*。这时，使用 SQL 的 GROUP BY 优化处理方法和 SubCube BY 子句指定的聚集函数逐个计算特定的 *cuboid*。

(4)合并步骤(3)计算的各 *cuboid*，形成结果关系，存储到磁盘。

从步骤(3)可以看出，Simple_Gcube 算法的核心是"使用 SQL 的 GROUP BY 优化处理方法计算 *cuboid*"。于是，实现 GROUP BY 操作的优化技术[2]都可应用于 Simple_Gcube 算法。

Simple_Gcube 算法可以进一步改进如下：

(1)我们可以在处理 SQL 查询段的同时处理 GCUBE 段。这样可以省略建立和读写临时关系 TMP 的时间。

(2)一个 Gcube 操作计算的各 *cuboid* 之间具有一种"导出"关系，即某些 *cuboid* 可以由另一些 *cuboid* 直接导出。Gcube 操作计算的 *cuboid* 集合在"导出"关系下构成一个代数格[4]。在步骤(3)处理 Gcube 操作时，我们可以利用这个代数格，采用新的高效率算法充分利用已经计算出的 *cuboid* 计算其余 *cuboid*，而不使用 SQL 的 GROUP BY 处理算法从原始关系逐个计算各 *cuboid*。我们可以首先建立 *cuboid* 集合的格，然后使用如下规则优化计算各 *cuboid*：

最小双亲规则：对于两个 *cuboid* C_i 和 C_j，若 C_i 可以由 C_j 计算出，并且 C_j 比 C_i 多一个属性，则称 C_j 为 C_i 的双亲，若 C_j 是 C_i 的所有双亲中最小的，则 C_j 称为 C_i 的最小双亲。最小双亲规则是指当计算一个 *cuboid* 时，使用其最小双亲。

主存优先规则：利用内存中已经计算出的 *cuboid* 导出其它的子 *cuboid*，以减少磁盘 I/O。

平摊扫描规则：在一个 *cuboid* 的一次扫描过程中计算尽可能多的其它 *cuboid*。

共享排序规则：针对基于 sort 的算法，使多个 *cuboid* 共享一次排序的结果。

共享划分规则：针对基于 hash 的算法，当 hash 表太大内存中放不下时，将数据进行划分，在每一个划分上计算尽可能多的聚集。

(3)如果原始关系已经分布在多个磁盘或多个处理结点上，我们可以使用并行算法提高 Gcube 操作的效率。

结论 本文认真分析了 Cube 操作，针对其局限性对该操作加以扩充，提出了更具一般性的 Gcube 操作，Gcube操作可以对多个关系的连接作

(下转第 45 页)

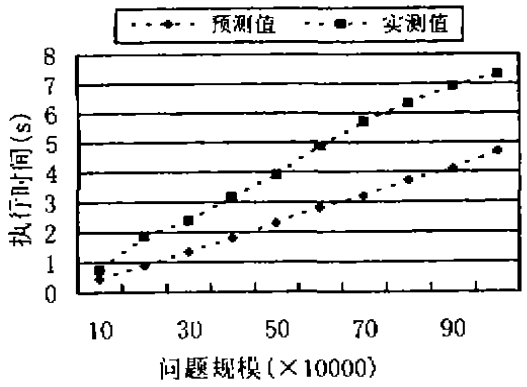

图 1 PSRS 预测值与实测值的比较图

信时间为 $O(n)$，计算时间为 $O(n\log n)$，但通信时间在目前的问题规模下，占了总时间的主要部分，用户负载对算法的性能影响并不明显。随问题规模的增大，总时间成线形增长趋势，与实测的趋势一致。从实验结果来看，预测值与实测值误差约为 40%，基本可以接受。

3.4 矩阵相乘算法的实验结果

按照同样的方法我们对矩阵相乘算法[6]进行分析，并与实际运行结果进行比较，如图 2。我们测量了四个进程在两个处理器上分布的两种情况，其一是计算速度较快的机器承担 3 个进程，另一台机器承担 1 个进程，其二是平均分布，在两台机器上各分布 2 个进程。实验结果表明误差在 20—30%左右，实测结果与预测结果反映的执行时间与问题规模大小的关系基本一致。

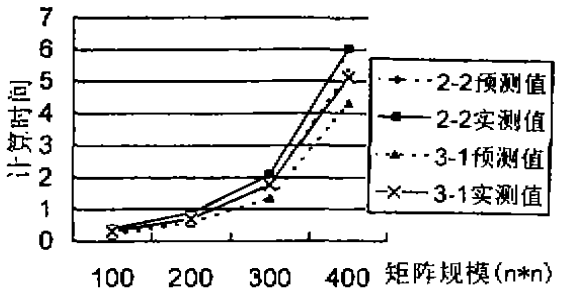

图 2 矩阵乘法实测值与预测值的比较

结束语 可见 NHBSP 并行计算模型考虑了 NOW 环境的两大特点，反映了用户负载和计算节点的异质性等对算法的影响。该模型对 PSRS 算法和矩阵相乘算法的分析结果与实测结果的比较，表明了该模型具有一定的可用性，性能预测也在可接受的准确度之内。

参 考 文 献

1 Message Passing Interface Forum. MPI: A message-passing interface standard. Intl. J. of Supercomputer Applications, 1994, 8(3/4)

2 陈国良 更实际的并行计算模型. 小型微型计算机系统, 1995, 16(2)

3 计永昶, 卜添, 陈国良 并行播送和求和算法在几种实际计算模型上的设计和分析. 中国科学技术大学学报, 1995, 26(2): 95～103

4 Valiant L G. A Bridging Model for Parallel Computation. CACM, 1990, 33(8): 103～111

5 Yan Yong, et al. An effective and practical performance prediction model for parallel computing on nondedicated heterogeneous NOW. Journal of Parallel and Distributed Computing, 1996, 38: 63～80

6 计永昶, 卜添, 陈国良. 网络计算环境下并行算法及其可扩放性分析. 计算机研究与发展, 1997, (11): 844～849

(上接第 25 页)

Cube；允许对操作关系进行选择操作，在满足条件的原始关系的子集合上执行 Gcube 操作；既可以计算对应于 GCUBE BY 属性集合的所有 *cuboid*，也可以选择计算特定的 *cuboid* 集合，而忽略其他的 *cuboid*，以提高效率；既可以在计算所有 *cuboid* 时使用相同的聚集函数集合，也可以使不同的 *cuboid* 使用不同的聚集函数集合。

本文还提出了一个实现 Gcube 操作的简单算法，并研究了提高 Gcube 操作效率的优化技术。我们将进一步深入研究高效率实现 Gcube 操作的算法。

参 考 文 献

1 Agrawal R, et al. Modeling Multidimensional Databases. In: Proc. of the 13th Int'l Conference on Data Engineering. Birmingham, U. K., 1997. 105～116

2 Gray J, et al. Data Cube: A Relational Operator Generalizing Group-By, Cross-Tab and Sub-Totals. In: Proc. of the 12th Int. Conf. on Data Engineering, 1996. 152～159

3 Agarwal S, et al. On the Computation of Multidimensional Aggregates. In: Proc. of the 22nd VLDB Conference. Mumbai, India, 1996. 506～521

4 Harinarayan V, et al. Implementing Data Cubes Efficiently. In: Proc. of ACM SIGMOD Conf., 1996. 205～216